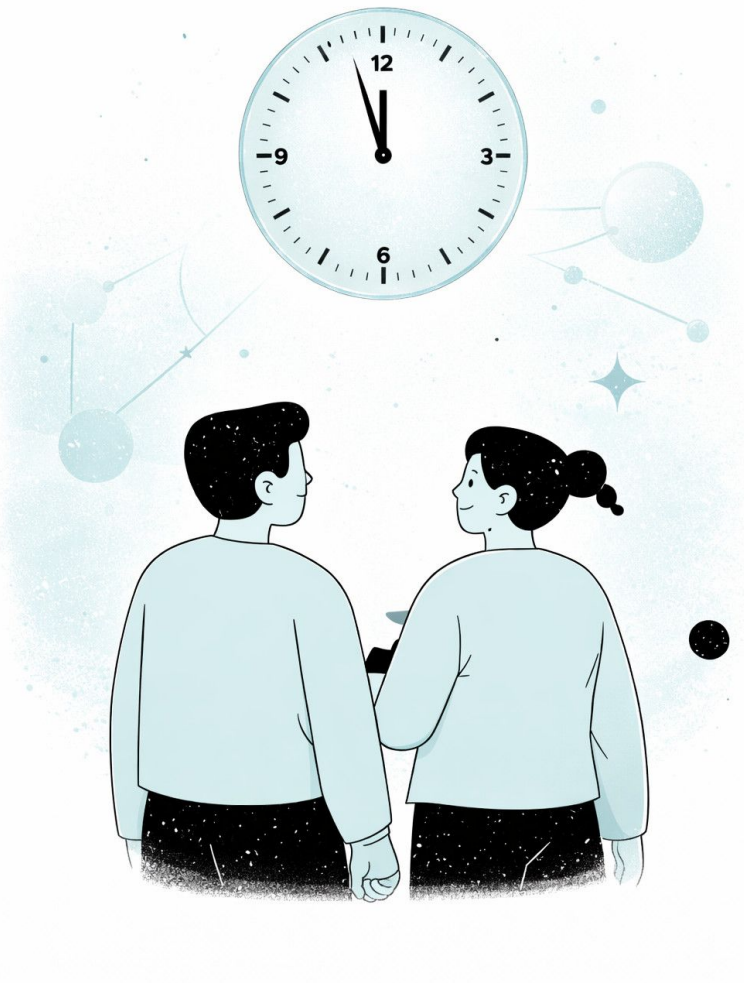
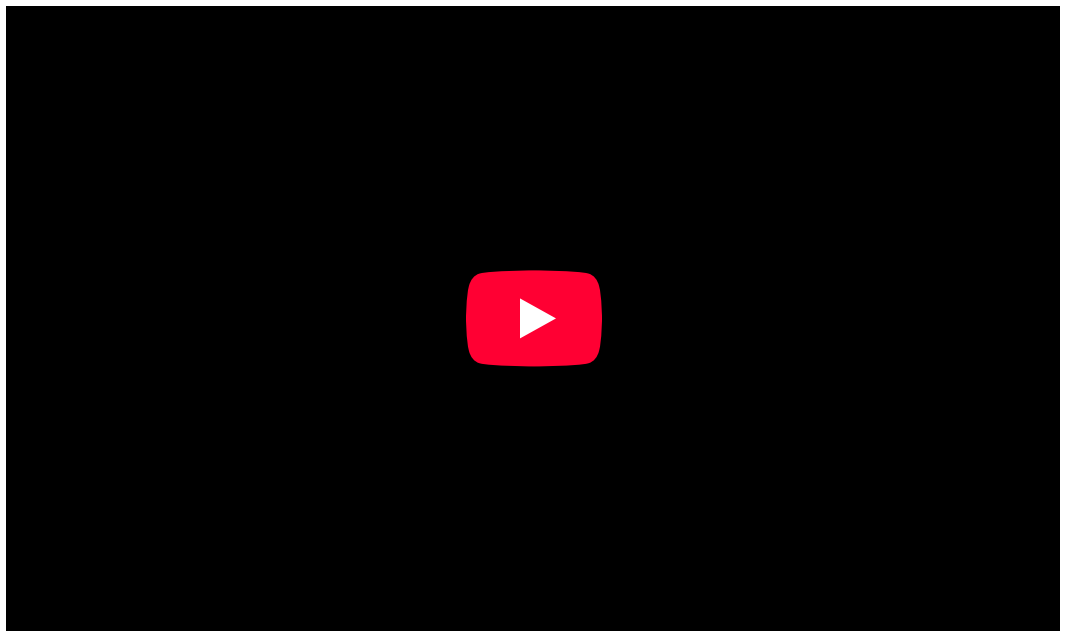
LESSON 08
End Game

Start



LESSON GOALS



Video Link

🎯 Goal Summary:

- Implement synchronized game timer
- End the game when time runs out
- Display final score screen
- Transition to RELOAD state
- Reset players, balls, teams, HUD
- Return to ORGANIZATION for a new match

Next

Learning Objectives + Deliverables

Learning Objectives

Create and synchronize a countdown timer

Stop gameplay at time expiration

Calculate WIN / LOSE / DRAW based on teams

Display final score UI

Reset state to ORGANIZATION

Deliverables

✓ Timer visible in HUD

✓ GAME_OVER transition at time end

✓ Display final score panel

✓ Game State cleanup

✓ Full return to ORGANIZATION state

GENERAL DESCRIPTION OF THE LESSON

High-Level Description

🎯 **Goal:** The game will end after a period of time. We should report to all the players when the game is over, show the final score and automatically reset the environment to start a new game.

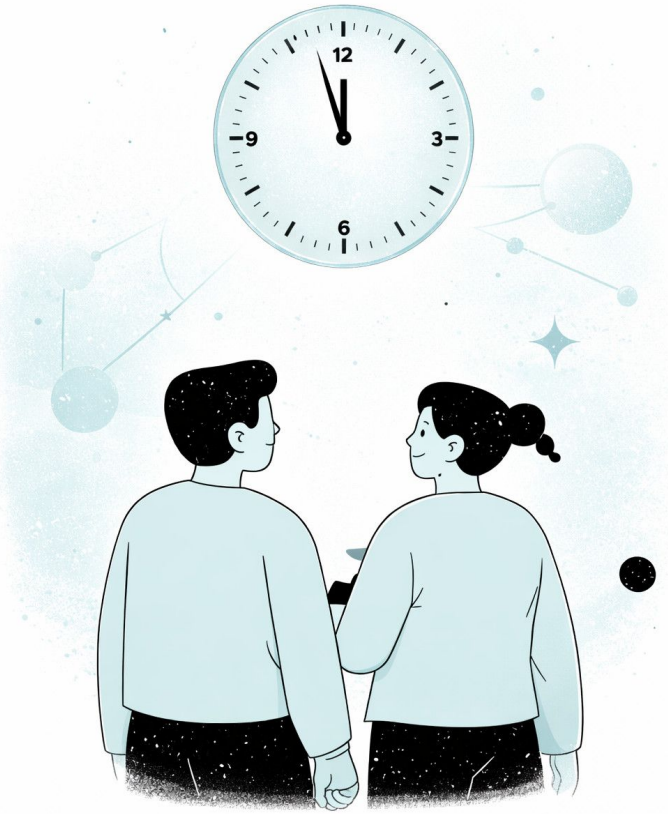
🏐 LEVEL 1: Advanced Challenge

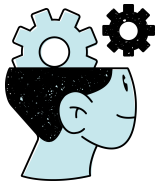
In this lesson we will:

- Timer counts during GAME state
- When timer reaches zero → GAME_OVER
- HUD shows WIN / LOSE / DRAW
- Despawn balls & indicators
- Teleport players into center of the area
- Reset all values:
  - Scores
  - Team indexes
  - HUD
  - Ball states
- Transition back to ORGANIZATION

🧠AI Lesson Prompt: End Game

Next





Exercise 1: The game will end after a defined period of time (5 minutes). Then a screen in the GameHUD will be displayed reporting the final result and informing if the team has won or lost.

- Actions:

1) In (**GameController**) we will implement these variables members and a constant:

2) Still in (**GameController**) implement function (**int GetRemainingTime()**)

3) In (**void Update()**) method, for the state GAME, implement the logic of time:
  - All the clients udpate (**timeGameProgress**) and the display in GameHUD
  - Only the Master can update (**timeProgressNetwork**)
  - Only the Master can run the transition of the state to GAME_OVER

4) In the script (**GameHUD**) implement create a new textfield in the gameObject to display the time and a public function (**void SetTime(int seconds)**) to set the value.

5) Back in the (**GameController**) script, use (**GameHUD::SetTime**) to update the time of the game

6) Test in the Unity Editor with a reduced amout of total time (10 seconds)

7) Repeat the test with multiple VRChat instances to verify that all clients are synchronized and the transition to state GAME_OVER is performed.

`Code Checkpoint: End Game Basic`

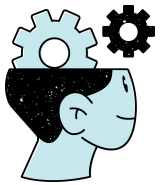
```
public class GameHUD : UdonSharpBehaviour
{

    [SerializeField]
    private TMP_Text time;

    public void SetTime(int seconds) {
       time.text = GetFormattedTimeMinutes(seconds);
    }

    private string GetFormattedTimeMinutes(int time) {
       int minutes = (int)time / 60;
       int seconds = (int)time % 60;

       // SECONDS
       string secondsBuf;
       if (seconds < 10) {
          secondsBuf = "0" + seconds;
       }
       else {
          secondsBuf = "" + seconds;
       }
       // MINUTES
       string minutesBuf;
       if (minutes < 10) {
          minutesBuf = "0" + minutes;
       }
       else {
          minutesBuf = "" + minutes;
       }
       return (minutesBuf + ":" + secondsBuf);
    }
```

```
public class GameController : UdonSharpBehaviour
{
 private const int TotalGameTime = 300; // Total time of the game

 [UdonSynced] private float _timeProgressNetwork = 0; // Network time for sync purposes

 private float _timeGameProgress = 0; // Current time progressed

 // OTHER CODE ...
```

```
public class GameController : UdonSharpBehaviour
{
    void Update()
    {
        switch (_currentState)
        {
          case GameState.GAME:
              _timeGameProgress += Time.deltaTime;
              if (Networking.IsMaster)
              {
                  _timeProgressNetwork = _timeGameProgress;
              }
              if (_timeGameProgress > TotalGameTime)
              {
                  if (Networking.IsMaster)
                  {
                      SetState(GameState.GAME_OVER);
                  }
              }
              else
              {
                  CheckPlayersCollision();
              }
              break;

     // OTHER CODE ...
```

```
public int GetRemainingTime()
{
    return (int)(TotalGameTime - _timeGameProgress);
}
```

```
public class GameController : UdonSharpBehaviour
{
    void Update()
    {
        switch (_currentState)
        {
          case GameState.GAME:
              _timeGameProgress += Time.deltaTime;
              gameHUD.SetTime(GetRemainingTime());
              if (Networking.IsMaster)
              {
                  _timeProgressNetwork = _timeGameProgress;
              }
              if (_timeGameProgress > TotalGameTime)
              {
                  if (Networking.IsMaster)
                  {
                      SetState(GameState.GAME_OVER);
                  }
              }
              else
              {
                  CheckPlayersCollision();
              }
              break;

      // OTHER CODE ...
```



Exercise 2: Create in the GameHUD a new panel that will inform the player the game is over, if he has won or lost and the final score.

- Actions:

1) In (**GameHUD**) create these variables members and the respective GameObjects in the GameHUD in scene:

2) Still in (**GameHUD**) create the public method **(void ShowGameOver(GameResult result, int scoreRed, int scoreBlue))** that will hide the panelGame and show the panelFinalScore with the final result. Use the enum:

3) In (**GameController**) when changing to GAME_OVER state, calculate the game's result (VICTORY, DEFEAT, DRAW) of the local player and call (**GameHUD::ShowGameOver**) to show the final score

4) Use the change to state RELOAD to make disappear the panel with the final score after 5 seconds

5) In the RELOAD state, you will have to (**DespawnBalls**) all the balls, (**DespawnAllIdentificationBalls**) and (**ResetPlayerIndexes**) for the next team formation.

6) You can test in the Unity editor the VICTORY and DRAW results

7) Test it with multiple VRChat instances to verify the system is working.

`Code Checkpoint: End Game Final Score`

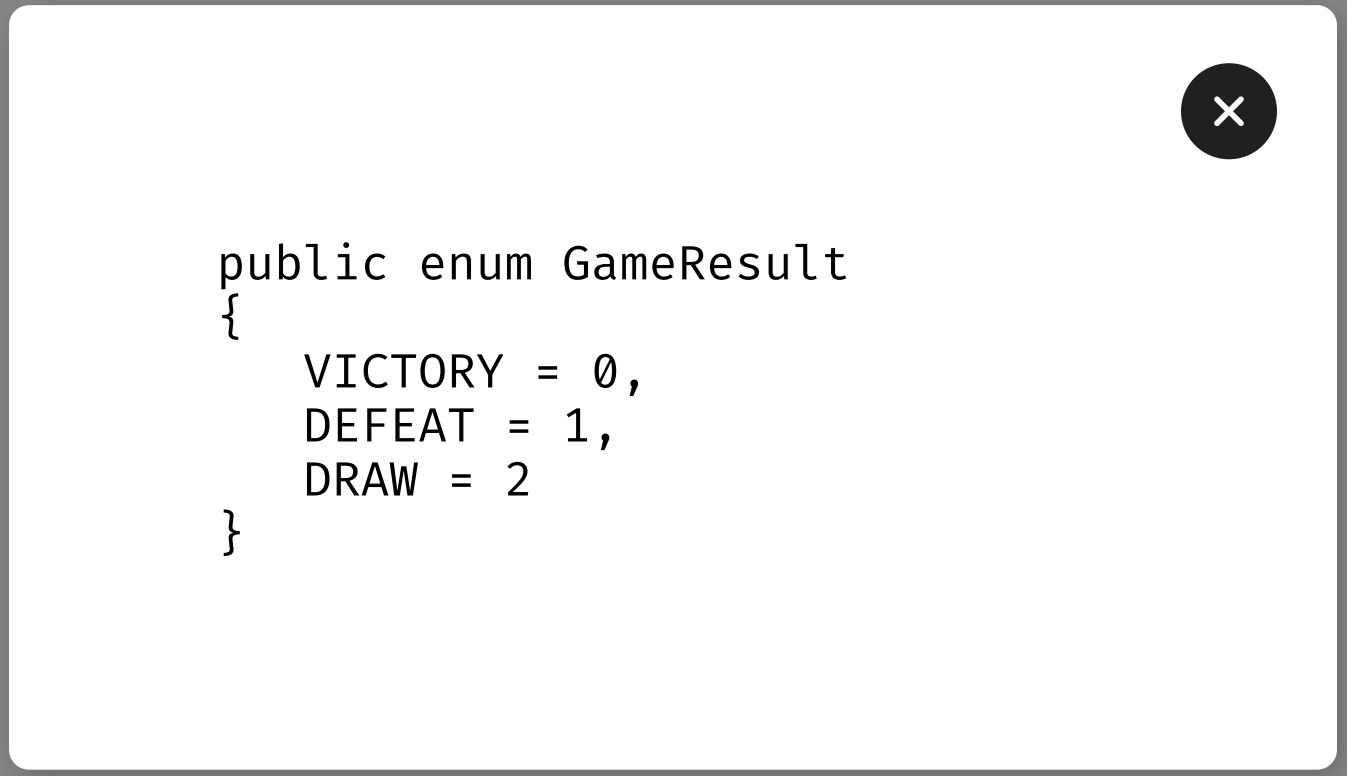
```
public class GameHUD : UdonSharpBehaviour
{
    // OTHER CODE ...

    public void ShowGameOver(GameResult result, int scoreRed, int scoreBlue)
    {
        panelGame.SetActive(false);
        panelFinalScore.SetActive(true);
        finalScoreTeamRed.text = scoreRed.ToString();
        finalScoreTeamBlue.text = scoreBlue.ToString();
        switch (result)
        {
            case GameResult.VICTORY:
                finalTextInformation.text = "VICTORY!!!";
                break;
            case GameResult.DEFEAT:
                finalTextInformation.text = "DEFEAT";
                break;
            case GameResult.DRAW:
              finalTextInformation.text = "DRAW";
              break;
         }
     }
```

```
public class GameHUD : UdonSharpBehaviour
{

    [SerializeField]
    private GameObject panelGame; // Container of the main components

    [SerializeField]
    private GameObject panelFinalScore; // Container of final score elements

    [SerializeField]
    private TMP_Text finalTextInformation; // "WIN", "LOSE", or "DRAW"

    [SerializeField]
    private TMP_Text finalScoreTeamRed; // The final score for red team

    [SerializeField]
    private TMP_Text finalScoreTeamBlue; // The final score for blue team
```


Hierarchy
All
MultiBallRugby*
VRCWorld
Main Camera
Directional Light
Floor
EventSystem
ScoreController
GameController
BallsController
GrantStadium
GameHUD
Game
Background
Team
Score
Time
Development
FinalScore
Image
Information
IconBlue
IconRed
ScoreBlue
ScoreRed
SoundsController
Scene
Game
Project Settings
Center
Local
2D
Your are RED
0
0
00:00
You Won!
99
99

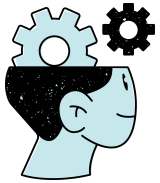
```
public class GameController : UdonSharpBehaviour
{
    // OTHER CODE ...

    private void StateChanged()
        ....
        case GameState.GAME_OVER:
            gameHUD.SetState("GAME_OVER");
            Team playerTeam = GetPlayerTeam(Networking.LocalPlayer.playerId);
            GameResult gameResult = GameResult.DEFEAT;
            if ((playerTeam == Team.TEAM_RED) && (scoreController.GetScoreTeamRed() > scoreController.GetScoreTeamBlue()))
            {
                gameResult = GameResult.VICTORY;
            }
            if ((playerTeam == Team.TEAM_BLUE) && (scoreController.GetScoreTeamBlue() > scoreController.GetScoreTeamRed()))
            {
                gameResult = GameResult.VICTORY;
            }
            if (scoreController.GetScoreTeamBlue() == scoreController.GetScoreTeamRed())
            {
                gameResult = GameResult.DRAW;
            }
            gameHUD.ShowGameOver(gameResult, scoreController.GetScoreTeamRed(), scoreController.GetScoreTeamBlue());
            break;
```

```
public class GameController  : UdonSharpBehaviour
    ...
    void StateChanged()
        _timeGameProgress = 0:
        ...
        case GameState.RELOAD:
            gameHUD.ShowGameHUD(); // Restore GameHUD visibility
            ...

    void Update() {
        ...
        case GameState.GAME_OVER:
           _timeGameProgress += Time.deltaTime;
           if (_timeGameProgress > 5) // Transition to reload state
           {
              if (Networking.IsMaster) {
                     SetState(GameState.RELOAD);
              }
           }
           break;

public class GameHUD : UdonSharpBehaviour
                    …
     public void ShowGameHUD()
     {
        panelGame.SetActive(true);
        panelFinalScore.SetActive(false);
     }
```



```
public enum GameResult
{
    VICTORY = 0,
    DEFEAT = 1,
    DRAW = 2
}
```

```
public class GameController  : UdonSharpBehaviour
    ...
     void StateChanged()
         ...
          case GameState.RELOAD:
             gameHUD.ShowGameHUD();
             ballController.DespawnBalls();
             DespawnAllIdentificationBalls();
             ResetPlayerIndexes();
             ...
public class BallController  : UdonSharpBehaviour
{    ...
     public void DespawnBalls()
     {
        foreach (GameBall ball in balls)
        {
           ball.DespawnBall();
        }
     }
     ...
```



Exercise 3: We are going to reload the game and organize a new game. The players will respawn close to the center of the game field facing the center with the button to start the game ready for them to press.

- Actions:

1) In (**GameController**), program the transition after 1 second from RELOAD state to the ORGANIZATION state.

2) Create a invisible respawn gameObject that you will use to teleport all the players when we change to the RELOAD state. In the (GameController) script, create the variable member and initialize it:

3) When changing to the RELOAD state you should teleport the players to the previously created respawn position and face them towards the center of the field and reset all the state (number of players, ScoreController, GameHUD).

4) Still in the RELOAD state, the last action we will perform is to call a function named (**ReAssignExistingPlayers**) that will retrieve all the existing players and will perform (**AssignNewPlayer**) for each one of them. This operation requires of some **refactoring** in the process of (**OnPlayerJoined(VRCPlayerApi player)**), so feel free to ask the AI.

5) Test in the Unity editor. You should be able to restart the game.

6) Test with multiple VRChat instances (2 and 4)

`Code Checkpoint: End Game Reload`

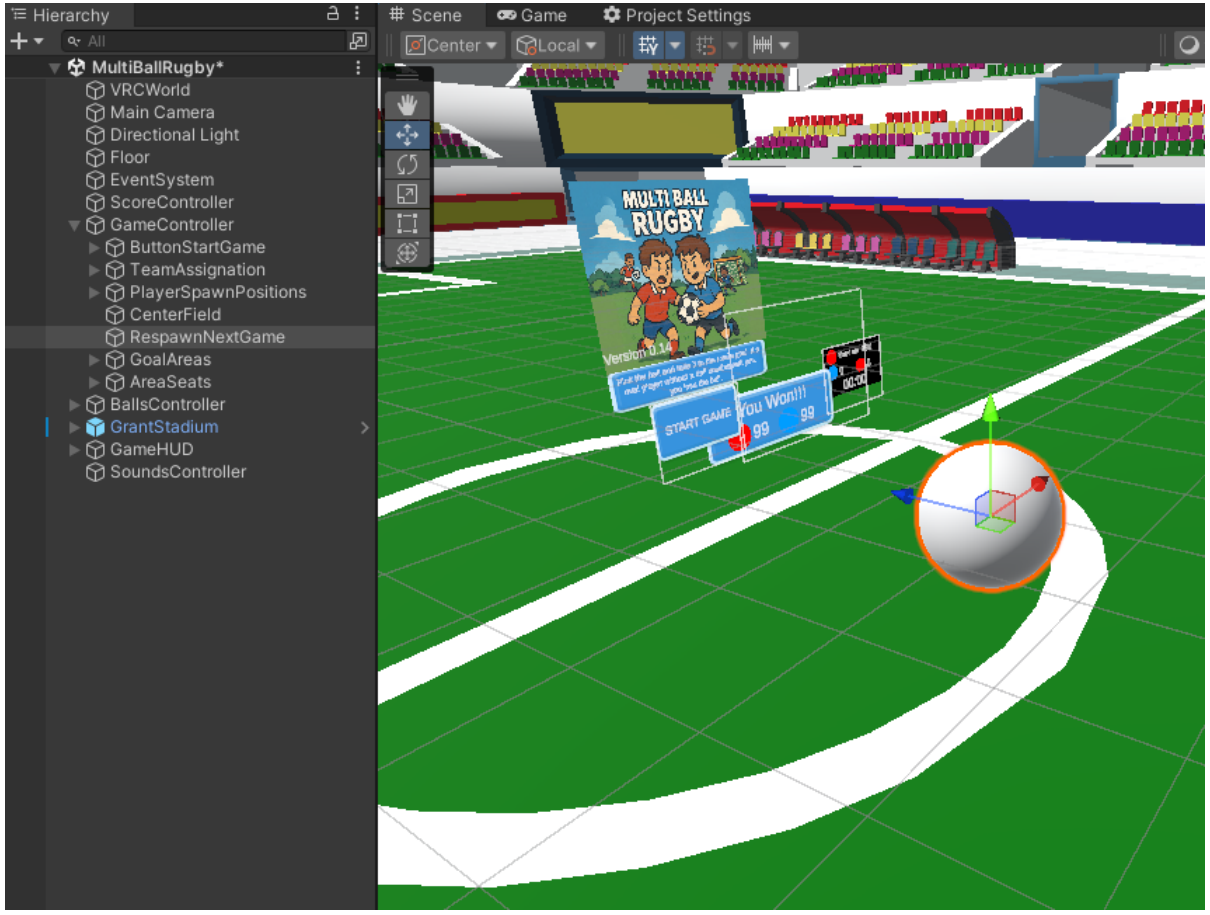
```
public class GameController : UdonSharpBehaviour
{
    [SerializeField]
    private Transform respawnInField;
```

```
public class GameController : UdonSharpBehaviour
{
     void Update() {
          ...
          case GameState.RELOAD:
               _timeGameProgress += Time.deltaTime;
               if (_timeGameProgress > 1)
               {
                    if (Networking.IsMaster)
                    {
                         SetState(GameState.ORGANIZATION);
                    }
               }
               break;
```

```
public void SetState(GameState newState)
{
    if (Networking.IsMaster)
    {
        syncedState = (int)newState;
        _currentState = (GameState)syncedState;
        RequestSerialization();
        StateChanged();
    }
}
```

```
public class GameController : UdonSharpBehaviour
{
  public override void OnPlayerJoined(VRCPlayerApi player) {
     if (player == Networking.LocalPlayer) {
        _currentState = GameState.ORGANIZATION;
        StateChanged();
     }
     if (Networking.IsMaster) {
         if (_currentState == GameState.ORGANIZATION) {
            AssignNewPlayer(player, true);
         }
     }
  }
  private void AssignNewPlayer(VRCPlayerApi player, bool updateButtonState) {
     if (_totalPlayersInGame < MaxPlayersPerTeam * 2) {
        bool isInTeamBlue = AssignPlayerToTeam(player);
        _totalPlayersInGame++;
        SpawnIdentificationBall(player, isInTeamBlue);
        if (updateButtonState) {
           RequestSerialization();
           buttonStartGame.NetworkEnable(_totalPlayersInGame % 2 == 0);
        }
     }
  }
  private void ReAssignExistingPlayers() {
     if (Networking.IsMaster) {
        _totalPlayersInGame = 0;
        VRCPlayerApi[] playersToReAssign = new VRCPlayerApi[100];
        VRCPlayerApi.GetPlayers(playersToReAssign);
        int count = VRCPlayerApi.GetPlayerCount();
        for (int i = 0; i < count; i++) {
           AssignNewPlayer(playersToReAssign[i], false);
        }
        RequestSerialization();
        buttonStartGame.NetworkEnable(_totalPlayersInGame % 2 == 0);
     }
  }
  private void StateChanged() {  ...

     switch (_currentState) {
        case GameState.ORGANIZATION:
           gameHUD.SetState("ORGANIZATION");
           gameHUD.ShowGameHUD();
           buttonStartGame.ShowStart();
           break;
```

```
public class GameBall: UdonSharpBehaviour
{
  public void DespawnBall()
  {
      if (isAlive)
      {
          if (Networking.IsOwner(gameObject))
          {
              isAlive = false;
              ownerPlayerId = -1;
              this.transform.position = _resetPosition;
              _hasBeenRespawned = true;
              impulseVector = Vector3.zero;
              ActivatePhysics(false);
              RequestSerialization();
              OnDeserialization();
          }
      }
    }
    private void Update()
    {
        if (!isAlive)
        {
            transform.position = _resetPosition;
        }
        else
        if (_currentOwner != null)
        {
            ...
```


Hierarchy
Scene
Game
Project Settings
MultiBallRugby*
VRCWorld
Main Camera
Directional Light
Floor
EventSystem
ScoreController
GameController
ButtonStartGame
TeamAssignation
PlayerSpawnPositions
CenterField
RespawnNextGame
GoalAreas
AreaSeats
BallsController
GrantStadium
GameHUD
SoundsController
MULTI BALL RUGBY
START GAME
You Won!!!

```
public class GameController : UdonSharpBehaviour
{
  void StateChanged() {
      ...
     case GameState.RELOAD:
        Quaternion orientationCenter =
          Quaternion.LookRotation((centerField.transform.position - respawnInField.position).normalized, Vector3.up);

        Networking.LocalPlayer.TeleportTo(respawnInField.position, orientationCenter);
        _playersForTeam = 0;
        _totalPlayersInGame = 0;
        scoreController.ResetScore(); // Implement public method
        gameHUD.ResetHUD(); // Implement public method
        ReAssignExistingPlayers(); // Implemented in the next refactoring
        break;
```

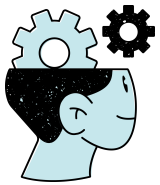
```
public class ScoreController : UdonSharpBehaviour
{
   ...
  public void ResetScore()
  {
    scoreTeamBlue = 0;
    scoreTeamRed = 0;
  }
```

```
public class GameHUD : UdonSharpBehaviour
{
     ...
    public void ResetHUD()
    {
        ShowGameHUD();
        scoreBlue.text = "0";
        scoreRed.text = "0";
    }
```

LESSON 08 COMPLETED

You now have:

- A synchronized game timer
- Automatic GAME_OVER transition
- Final score UI
- Clean reset logic
- Full replayable loop



Self-Evaluation

It’s time to put what we’ve learned into practice! Here are 4 questions to check by yourself what you have learnt in this lesson.

- Question 1
- Question 2
- Question 3
- Question 4

Why is the game timer updated and controlled by the Master player?

- To ensure a single authoritative time source for all players
- Because non-Master players cannot use Time.deltaTime
- Because only the Master can see the HUD timer

What is the main purpose of the GAME_OVER state?

To disconnect all players from the world

To stop gameplay and display the final results of the match

To immediately restart the match without showing results

Why are balls and team indicators despawned during the GAME_OVER / RELOAD phase?

To prevent players from seeing the final score

To reset the game to a clean and neutral state before the next match

To improve lighting performance

Why does the game return to the ORGANIZATION state after RELOAD?

To allow players to leave the world

To increase game duration

To prepare players for the next match with fresh teams and setup

Help us to improve

Did you understand how the game timer works and why it must be master-controlled?

Write your answer here.

Send

Was the GAME_OVER → RELOAD → ORGANIZATION flow clear?

Write your answer here.

Send

Did you struggle with HUD transitions or state changes?

Write your answer here.

Send

What could make the end-game flow easier to understand or visualize?

Write your answer here.

Send